MIGRACIÓN DE LA APLICACIÓN MÓVIL DE ALEGRA A IONIC FRAMEWORK 3

MATEO CALLE MOLINA

Trabajo de grado para optar al título de

Ingeniero de sistemas y computación

Santiago Villegas Giraldo, CTO Soluciones Alegra S.A.S.

UNIVERSIDAD EIA
SOLUCIONES ALEGRA S.A.S.
INGENIERIA DE SISTEMAS Y COMPUTACIÓN
ENVIGADO

2017

CONTENIDO

pág.

LISTA DE TABLAS

LISTA DE FIGURAS

GLOSARIO

Alegra: aplicación de facturación y contabilidad para pymes, creada y mantenida por Soluciones Alegra S.A.S.

PYMES: acrónimo para “pequeñas y medianas empresas”.

ECMAScript (o ES): básicamente es el nombre formal de JavaScript, estandarizado y regulado por el ente de control ECMA.

Angular: framework de JavaScript para hacer aplicaciones e interfaces interactivas. Cuando nos referimos a *Angular* nos referimos a la versión 2.x en adelante, mientras que cuando mencionamos *AngularJS* indicamos que es la versión 1.x.

Ionic: framework basado en Angular para hacer aplicaciones híbridas.

RESUMEN

La empresa *Soluciones Alegra S.A.S.* tiene una aplicación web llamada *Alegra*, un software de facturación y contabilidad para pymes. Esta esta diseñada para funcionar en todos los navegadores modernos, sin embargo no está optimizada para smartphones. Es por esto que se tiene una aplicación móvil disponible en dispositivos iOS y Android como producto complementario a su software en la nube. Aunque la aplicación ha tenido buena acogida entre sus usuarios, se desea mejorar la experiencia de aquellos que la usan y de los nuevos que se registran mediante este medio, además de darle un poco más de estabilidad al desarrollo. Es por esto que se plantea migrar la aplicación móvil, que está hecha en el (framework híbrido) Ionic versión 1.2, a la última versión (al día de hoy, 3.6.0), la cual trae muchas mejoras de rendimiento y de interfaz gráfica, dándole así una experiencia más fluida y nativa.

Durante la realización de este proyecto, hubo varias etapas para poder llegar al objetivos final. En primera instancia se escogieron las tecnologías bases de en las que se desarrollaría el prototipo, como el lenguaje de programación, el lenguaje de hojas de estilo, así como la herramienta para automatización de tareas y transformación de código. Posteriormente se diseñó la estructura base de la aplicación, cómo se iban a organizar y nombrar todos los archivos y carpetas, y cómo debía estar constituida cada funcionalidad. Luego, como la fase más importante y extensiva, se procedió a implementar el prototipo en sí, distribuyendo el desarrollo de cada funcionalidad en sprints de un mes cada uno. Y finalmente, se hizo una breve comparación de rendimiento entre el modelo elaborado y el disponible actualmente en las tiendas.

Como resultados de este proyecto se creó un prototipo de la aplicación móvil de *Alegra* hecho en Ionic 3.6.0 con las funcionalidades básicas de la aplicación actual (facturas, gastos, contactos y productos/servicios) con la cuál se espera obtener amplias ganancias en rendimiento, aumentar la mantenibilidad y estabilidad de la aplicación, incrementar la velocidad y facilidad de desarrollo, y por encima de todo, ofrecer una mejor experiencia de usuario.

Palabras clave: Ionic, Angular, aplicación, móvil, híbrida

ABSTRACT

The company *Soluciones Alegra S.A.S.*, has a web application called *Alegra*, it's an invoicing and accounting software for small and medium businesses. It is designed to work in every modern web browser, nonetheless, it is not optimized for smartphones. For this reason, there's and mobile application available for iOS and Android as a complimentary product to its cloud software. Even though the application has been very well received among *Alegra's* users, there's a desire to improve the user experience of those who use it and of those who register through this medium. And also bringing more stability to the codebase and the development cycle. For this reason, it has been proposed to migrate the mobile application, that's made in (the hybrid framework) Ionic 1.2, to the last version (3.6.0 as of today), which brings many performance and graphic interface improvements, giving a much more fluid and native experience.

During the making of this project, there where multiple stages towards achieving the final goal. In first instance, the base technologies in which the prototype was to be developed where chosen. Technologies such as the programming language, the cascading stylesheets language and a tool for bundling, task automation and code transformation. Subsequently, the application structure was designed, how all the files and folders where to be named and organized and how each functionality was supposed to be made. Later, as the most important and extensive phase, it was proceeded to implement the prototype, distributing the development of each function throughout a series of sprints of one month each. And finally, a brief performance comparison between the elaborated model and the currently available in the marketplaces was made.

As results of this project, a prototype of the mobile application of *Alegra* was created using Ionic 3.6.0 with the basic functionalities (invoicing, expenses, contacts and products/services) of the current app in the stores, in which it is expected to gain large performance profits, increase the maintainability and stability of the codebase, boost the ease and speed of the development cycle, and, above all, offer a much better user experience.

Keywords: Ionic, Angular, hybrid, mobile, application

INTRODUCCIÓN

En este documento se hará una breve historia de las tecnologías que han permitido la creación de aplicaciones móviles híbridas y su evolución a través de los años. Se analizará por qué se han empezado a realizar cada vez más y se discutirán las ventajas y desventajas que poseen para poder compararlas con las demás alternativas que existen hoy en el mercado. También se observará cuál es su posición en la actualidad y se expondrá cuál es el estado de arte en este tema.

Se examinarán las diferencias entre AngularJS (1.x) y Angular (2+), así como también se hará para las versiones 1.x y 2+ de Ionic Framework. De esta forma se podrá comprender la decisión de migrar la aplicación móvil de Alegra a la última versión de Ionic. Posteriormente mostraremos el proceso por el cual se actualizó todo, se harán una breves comparaciones de rendimiento entre el prototipo y la aplicación móvil actual y se dejaran en evidencia las decisiones que se tuvieron que tomar en el camino y el porqué de cada una.

1. PRELIMINARES

1.1 CONTEXTUALIZACIÓN Y ANTECEDENTES

En el año 2014, el consumo de medios digitales en dispositivos móviles sobrepasó el consumo en computadores (Perez, 2014). Este era una pronóstico que se veía venir desde hace ya varios años pero que sólo se materializó recientemente. La importancia de tener soluciones móviles para suplir así esta demanda se hace cada vez más vital e importante para todo negocio que ofrezca soluciones basadas en la prestación de servicios via internet. Tanto es así que Google le llamó a estos momentos fugaces de interacción con los dispositivos móviles como micro-momentos (Google, s.f.).

Desarrollar aplicaciones móviles es complejo y tedioso, en parte porque requiere el conocimiento no solo de una sino de varias plataformas (iOS, Android), y junto con estas, distintos lenguajes de programación, distintas herramientas y distintas maneras de crear interfaces gráficas. Esto implica desarrollar y mantener diferentes versiones, tener varios equipos para cada una de las plataformas en las que se desee tener presencia. Es debido a estas problemáticas, y al hecho de que existe mayor cantidad de desarrolladores centrados en tecnologías web[1], que se crearon las aplicaciones híbridas. Aplicaciones que corren dentro de un WebView, una vista que muestra una página web (Google). Este tipo de aplicaciones, a pesar de no ser nativas pueden acceder a los componentes nativos del sistema operativo y del dispositivo (The Apache Software Foundation, s.f.).

Sin embargo, a pesar de que el concepto de aplicaciones híbridas es demasiado bueno y ha ido mejorando de forma incremental, no es el adecuado para todo tipo de aplicaciones, en especial aquellas que requieren alto rendimiento (Fischer, 2015). Por el mismo hecho de correr sobre un WebView, las transiciones se limitan a CSS (abreviación para *hojas de estilo en cascada* por su nombre en inglés, *Cascading Styles Sheets*), y no a las nativas de cada plataforma. Esto hace que en ciertos casos la aplicación se sienta lenta y deteriora así la experiencia de usuario.

Alegra, una aplicación web de administración y facturación en la nube para pymes, que ofrece su producto desde hace más de tres años en la web, decidió invertir recursos en la creación de una aplicación móvil no sólo para beneficiar a sus usuarios ya existentes, sino también para atraer una nueva audiencia a través de distintos medios (App Store y Play Store). Alegra tomó la decisión de ir por una solución híbrida debido a una serie de razones anteriormente mencionadas y previamente estudiadas a nivel interno. Y aunque

[1] Una búsqueda rápida el día 20 de mayo de 2016 en www.indeed.com arrojó 63.364 resultados de trabajos para “Web developer”, 7.206 para “iOS developer” y 7.311 para “Android developer”.

el resultado ha sido muy positivo[2], se puede mejorar, no sólo en la adición de nuevas funcionalidades para acercarse más a la experiencia de escritorio, sino en mejorar la experiencia del usuario. Es así como se llega a la pregunta, ¿cómo se puede mejorar el rendimiento y la experiencia de usuario de la aplicación híbrida de Alegra?

Entre los diferentes antecedentes tenemos:

- Sworkit, una aplicación de salud que provee distintos tipos de entrenamiento para perder peso, está hecha en Ionic. Ha sido descargada más de dos millones de (Lynch, Ionic's 2016 - By the Numbers, 2017) y tiene una calificación de 4.6 / 5 (Katie, The official Ionic blog, 2016) (Google).

- Dow Jones, desarrolló la aplicación MarketWatch, enfocada a quienes estén interesados en noticias y análisis de los mercados financieros. (Katie, Built with Ionic: MarketWatch, 2016).

- Recientemente, Microsoft sacó una aplicación de automatización de tareas hecha con Ionic. Decidieron ir por la opción híbrida debido a que les permitía reusar gran parte del código de su versión web (Yassour, 2017).

- Se han creado cerca de 4 millones de aplicaciones con Ionic desde que se lanzó en 2014 (Lynch, Ionic's 2016 - By the Numbers, 2017).

1.2 OBJETIVOS DEL PROYECTO

1.2.1 Objetivo General

Desarrollar un prototipo de aplicación móvil con las funcionalidades básicas de la aplicación móvil actual de Alegra usando la última versión de Ionic Framework.

1.2.2 Objetivos Específicos

- Elegir las tecnologías y herramientas que se usarán en el proyecto.
- Diseñar la estructura base de la aplicación al cuál se someterán cada una de las funcionalidades.
- Implementar cada una de las funcionalidades básicas de la aplicación actual.
- Realizar una comparación funcional entre la aplicación actual y el prototipo desarrollado.

[2] Puntuación de 4.3 en la Play Store (Google), accedido el 20 de mayo de 2016.

1.3 MARCO DE REFERENCIA

En el año 2009, salió un producto llamado PhoneGap, tal vez la primera herramienta que que le permitía a los desarrolladores crear aplicaciones híbridas, es decir, aplicaciones móviles usando tecnologías web (JavaScript, HTML, y CSS). Posteriormente, Adobe adquiere esta tecnología en el año 2011 (Shankland, 2011) y parte del código es donada como proyecto open source a Apache Foundation y se renombra como *Apache Cordova*, mientras que PhoneGap se mantiene como un producto ofrecido por Adobe, siendo Cordova la base.

Teniendo ya la base para el desarrollo de aplicaciones híbridas, empezaron a surgir muchos frameworks que encapsulaban Apache Cordova y construían sobre este, de modo que se concentraban más en tener una lista de componentes que se asimilaran visualmente a sus contrapartes nativas y tuvieran una experiencia fluida. OnsenUI, Framework7 y Famous son algunos de estos frameworks, pero Ionic fue el que más popular se volvió debido a su extensa librería de componentes y su rendimiento. Ionic, como ya dijimos, funciona sobre Apache Cordova y además también usa AngularJS para el desarrollo. AngularJS facilita muchísimo el desarrollo en JavaScript con funcionalidades como *two-way binding*. El problema es que para lograr esto, usan una técnica llamada *dirty checking*, que lo que hace es que cada que una propiedad cambia, AngularJS tiene que iterar por todas las variables para ver si alguna cambió y así actualizar la interfaz (Gundersen, A comparison of the two-way binding in AngularJS, EmberJS and KnockoutJS, 2013). De modo que cuando el modelo empieza a crecer, AngularJS empieza a sufrir en cuanto a rendimiento. Este es un sólo ejemplo de muchas otras malas decisiones que sufre esta primera iteración de AngularJS. Debido a esto, Google decidió reescribir el framework desde cero teniendo en cuenta todos los problemas anteriores. Tanto así que son prácticamente frameworks diferentes, lo único que conservan en común es el nombre.

Angular (2+) además de resolver todos los problemas que ya existían, desde un inicio se diseñó para funcionar alrededor de componentes, poniéndola a la par de nuevos frameworks más eficientes como React y Vue. El soporte móvil fue algo que siempre se tuvo en cuenta, es por esto que su nueva arquitectura separó la capa de aplicación, de la capa de renderizado, de modo que, en teoría, cualquier vista (HTML, XML, etc.) puede ser usada al momento de ejecución. El equipo de Ionic apoyó la decisión de Google y trabajaron muy de la mano con el equipo de Angular para sacar la nueva versión adelante. Y las ganancias son evidentes, las nuevas versiones de Ionic son mucho más rápidas, hay más componentes, que además cambian su apariencia según la plataforma en la que se encuentren, y permite hacer cosas que con Ionic 1.x eran supremamente difíciles, como tener pestañas y menú lateral al tiempo. La navegación se hizo desde cero, y ya no depende de la URL, sino que funciona como en una aplicación nativa, teniendo *stacks* de pantallas a las que se les puede ir añadiendo nuevas, o quitando (push & pull from stack).

A pesar de todo, Ionic (y Apache Cordova) sigue siendo un framework de aplicaciones híbridas, y corre sobre una instancia del navegador que trae el dispositivo móvil, normalmente conocido como una *WebView*. Por lo que su rendimiento, aunque sea

bastante decente, nunca será como el de una aplicación nativa. Recientemente han empezado a coger muchísima fuerza una serie de frameworks que no corren sobre el *WebView* sino que compilan directamente a código nativo, Java y Objective-C para Android y iOS respectivamente. Las opciones más populares son, React Native de Facebook que tiene a React como base, NativeScript de Progress que usa Angular (2+), y Weex de Alibaba que utiliza Vue. Esto no es una idea precisamente nueva, de hecho, se viene haciendo desde ya hace vario tiempo con Xamarin de Microsoft, donde se desarrolla en C# y compila al lenguaje nativo para ambas plataformas. La innovación más grande es que, a diferencia de ellos (Xamarin), estos nuevos frameworks permiten desarrollar en JavaScript y compilan al momento de ejecución, no antes de empaquetar la aplicación. Este es el estado del arte en cuanto a aplicaciones híbridas y se debería tener en cuenta para futuros proyectos.

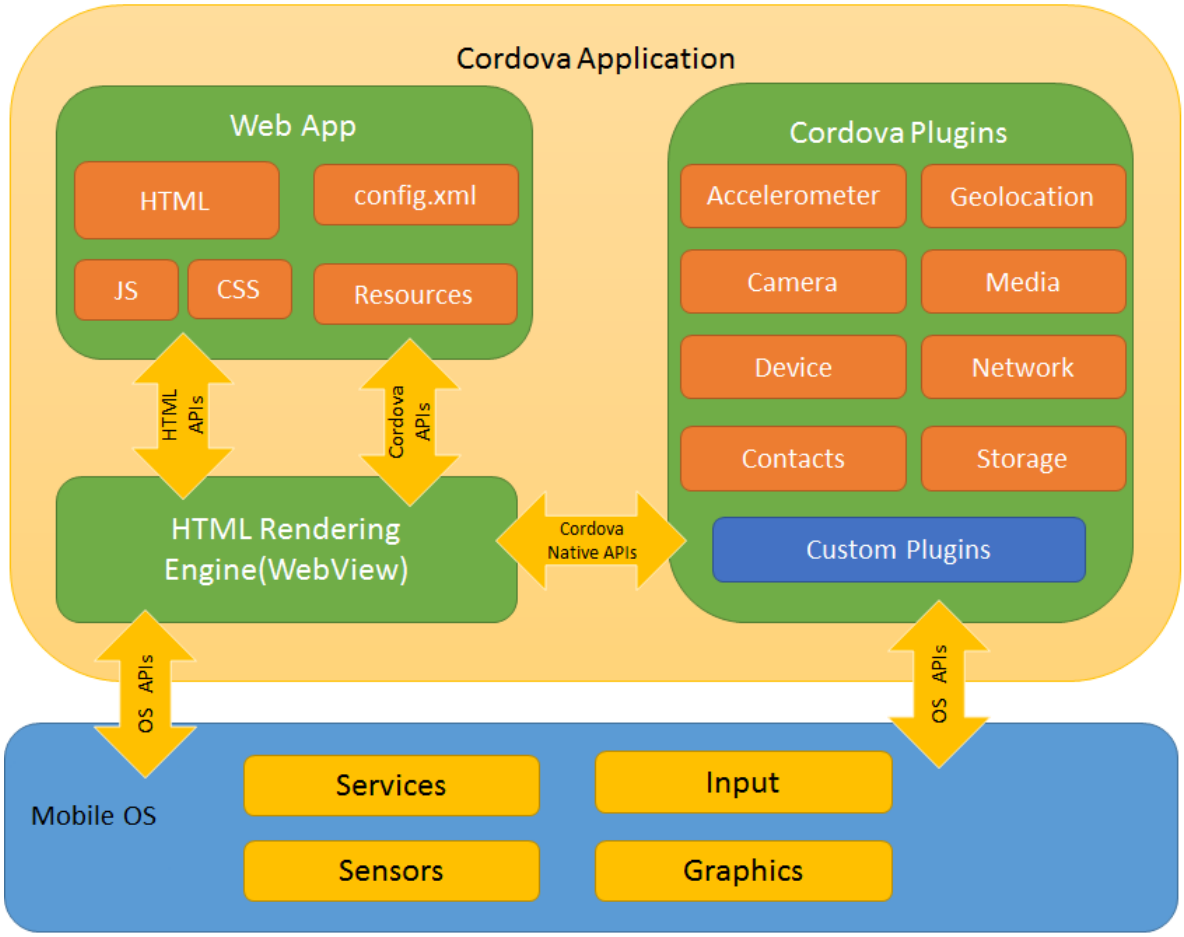


Figura 1. Arquitectura de una aplicación hecha en Apache Cordova Descriptive. Sacada de Apache Cordova Overview, n.d., Obtenida el 14de octubre, 2017 de https://cordova.apache.org/docs/en/latest/guide/overview/

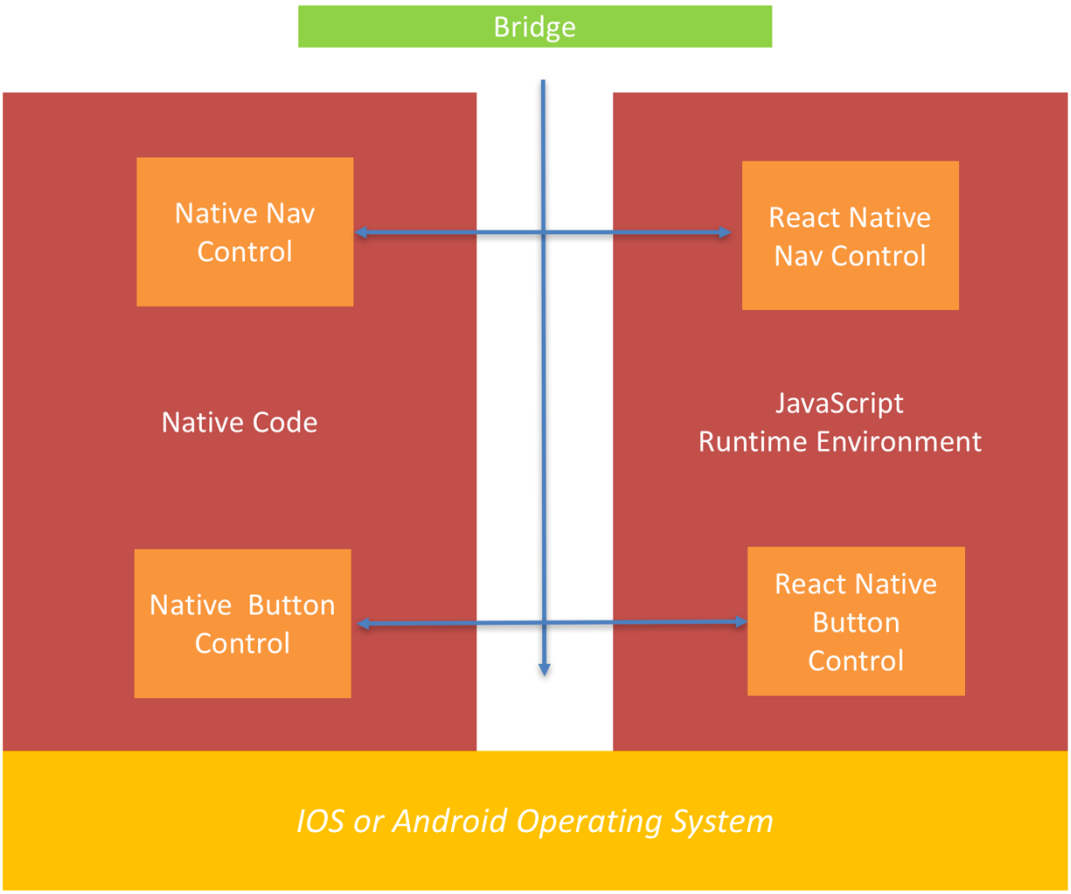


Figura 2. Arquitectura de una aplicación hecha en React Native. Tomado de React Native Architecture: Explained!, por P. Heard, Junio 3 de 2017, Recuperado el 14 de octubre, 2017 de https://www.logicroom.co/react-native-architecture-explained/

2. ENFOQUE Y METODOLOGÍA

2.1.1 Elección de las herramientas y tecnologías del proyecto

Antes de iniciar el proyecto, surgió la necesidad de definir ciertas herramientas y tecnologías que se usarían en el proyecto. Puntualmente hablamos del lenguaje de programación, del lenguaje de hojas de estilo y de la herramienta para automatización de tareas y transformación del código.

Lenguaje de programación:

Desde un inicio se definió que el prototipo se haría con Ionic, un framework para hacer aplicaciones híbridas, es por esto que ya se sabía que se tendría que usar un lenguaje que compilara o transformara a JavaScript. Las opciones que se tuvieron en cuenta son: ES5 (o Vanilla JavaScript), ES6, TypeScript y Dart.

ES5 quedó eliminada casi de inmediato debido a que es un estándar "viejo" que ya empieza a mostrar su edad, y aunque todos los navegadores lo entienden, las nuevas versiones tienen características que facilitan muchas cosas (*destructuring*, *arrow funtions*, *classes*, *string interpolation*, etc.).

Dart es un lenguaje de programación open source creado por Google en el año 2011 (Bak, 2011). Este posee su propia máquina virtual y compila a JavaScript. Google lo ha mantenido constantemente, y además lo usa internamente en proyectos tan grandes como AdWords. Otra ventaja muy importante para nosotros es que es soportado oficialmente por Angular. Sin embargo se decidió no usarlo debido a su baja popularidad y que es otro lenguaje el cuál hay que aprender.

Esto nos dejó con dos opciones, ES6 y TypeScript. El posterior es un *superset* de JavaScript creado por Microsoft y que además es open source. Estos dos son virtualmente iguales, la única diferencia a simple vista es que uno es tipificado (TypeScript) y el otro no (ES6). Todo código escrito en ES6 es TypeScript válido, pero no viceversa, y ambos son soportados oficialmente por Angular y Ionic. A fin de cuentas se escogió TypeScript debido a que trae más estabilidad al código, una mejor experiencia de desarrollo (con IntelliSense, como se puede observar en la Figura 3) y tal vez la razón más grande, es el lenguaje oficial tanto de Angular, como de Ionic, por lo que casi toda la documentación que se encuentra está hecha en TypeScript.

Figura 3. IntelliSense usando TypeScript

Lenguaje de hojas de estilo:

Para esta parte, aunque existen varias opciones, la decisión fue mucho más sencilla. Se necesitaba lenguaje cuya sintáxis fuera parecida a la de CSS pero con facilidades para escribir hojas de estilo como variables, funciones y estilos anidados. Las opciones más grandes eran Sass (.sass y .scss) y Less. Se definió usar Sass en su versión *.scss* debido a su alta popularidad, a que cualquier CSS válido es *.scss* válido y sobre todo porque ya se ha usado en otros proyectos en la empresa (app móvil actual y Alegra POS).

Estos son algunos ejemplos de las ventajas que se obtienen al usar Sass:

Variables:

```
// Sintaxis en SCSS

$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
```

```
  color: $primary-color;
}

// CSS compilado

body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Selectores anidados:

```
// Sintaxis en SCSS
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}

// CSS compilado
```

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav li {
  display: inline-block;
}

nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Herramienta para automatización de tareas y transformación de código:

La versión actual de la app móvil usa Gulp para compilar el código (ES6 a ES5), minificar, concatenar, añadir los prefijos de CSS, entre muchas otras cosas. Es una herramienta muy poderosa y extensible, pero que se ha quedado atrás en cuanto a funcionalidades con respecto a herramientas más nuevas. Webpack es un *"bundle manager",* que además de realizar todo lo anteriormente dicho, permite no sólo administrar el lenguaje de programación como tal, sino todas las dependencias que tenga internamente, como estilos e imágenes. El objetivo de Gulp es la automatización de tareas, por lo que toca indicarle manualmente qué debe hacer paso a paso, mientras que a Webpack sólo hay que pasarle una configuración y él te maneja todas las dependencias de forma automática. Se decidió usar Webpack porque es la herramienta oficialmente usada por el equipo de Ionic, y de hecho ya viene configurado con Ionic cuando uno crea un proyecto nuevo, por lo que no hubo que hacerle ningún cambio necesario. Además, ya se ha usado en otros productos de Alegra (POS).

2.1.2 Diseño de la estructura base la aplicación

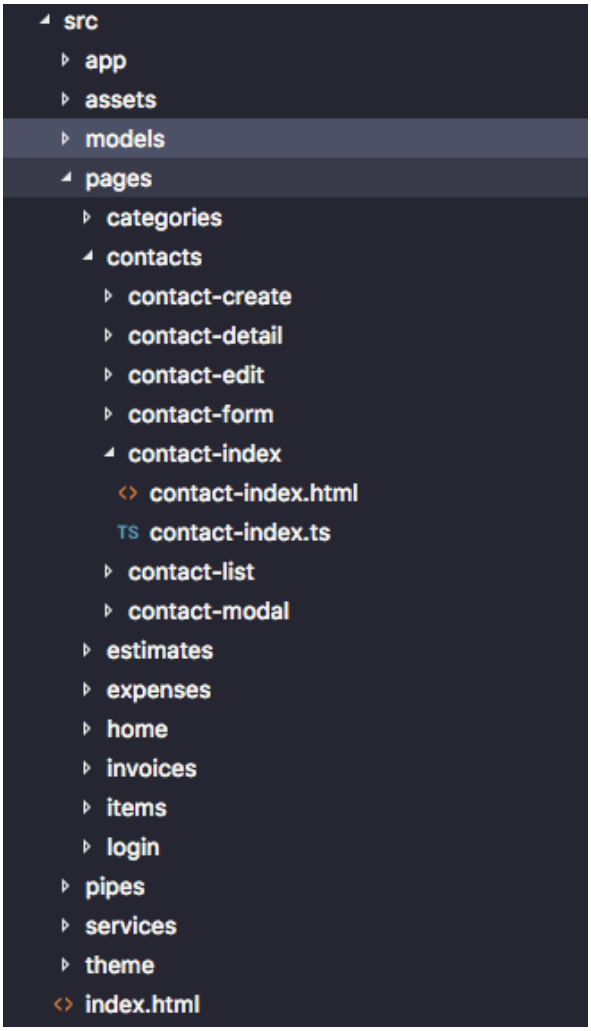


Figura 4. Estructura de la aplicación

Inicialmente se decidió seguir la guía de estilos oficial de Angular (Angular, 2017) de cómo estructurar una aplicación, convenciones de código, nombres, estilos, entre otros, sin embargo, hubo unos problemas con Ionic que impidió seguir la estructura proporcionada por la guía. Se quería separar cada funcionalidad en su propia carpeta autosuficiente, de modo que se incluyera sus páginas, sus componentes, servicios y modelos. El problema con esto es que todo lo que se usa en Angular se debe declarar para que la aplicación sepa de su existencia, y al separar la aplicación en diferentes módulos, se debe repetir la declaración de *IonicModule* en todos y cada uno de las funcionalidades debido a que esta es una pieza esencial para que Ionic funcione correctamente. Es así entonces, como se decidió usar una estructura más sencilla, y agrupar por tipo de archivo (*pages*, *models*, *services*, *pipes*, etc.). La única excepción es que cada componente tiene dos archivos, *component.ts* y *component.html*, agrupando la lógica y estructura en un mismo lugar, y si

hubiera alguno que tuviera estilos propios sólo para ese componente, también se pondrían ahí (*component.scss*).

Teniendo definida la estructura de los archivos, se prosiguió a definir la estructura de cada funcionalidad. Tomaremos todo lo concerniente a los contactos como ejemplo. Todas las páginas y componentes de una misma función deben ir bajo una carpeta que las agrupe. Como se puede ver en la Figura 4, todas las acciones concernientes a los contactos (index, detail, create, etc.) están dentro de la carpeta *contacts.* Cada funcionalidad que interactúe con Alegra, debe tener tanto un modelo (*contact.model.ts* para validación de tipos) como un servicio (*contact.service.ts*), ambos bajo sus respectivas carpetas. También se debe tratar de maximizar la reutilización de componentes, de modo que siempre que se encuentre un patrón que se repita, se sugiere fuertemente crear un nuevo componente para ser usado en las diferentes partes en las que se reproduce ese mismo comportamiento. Un ejemplo muy claro de esto son todos los componentes que Ionic brinda dentro de su framework, botones, listas, emergentes, notificaciones, entre muchos más. Todos estos son modelos que constantemente se están repitiendo y se pueden extraer a su propio componente, con su lógica y estilos.

2.1.3 Implementación de las funcionalidades

Siguiendo la metodología de desarrollo usada al interior de Alegra, se usó la metodología ágil SCRUM para la implementación del prototipo, tomando grandes libertades y cambios en dicha dinámica. Debido a que se era una sola persona, y no había forma de que se distribuyeran los papeles de SCRUM Master y desarrollador, no se llevaron a cabo reuniones diarias. Lo que si se hizo fue separar el desarrollo del prototipo en sprints de un mes cada uno, con el objetivo de que se hiciera cada gran funcionalidad en un sprint independiente.

A continuación, se muestran los avances en cada sprint y algunos de los retos que se encontraron en el camino.

Sprint 1:

- Estructura básica inicial
- Menú lateral (Figura 5)
- Index de contactos (Figura 6)
- Detalle de contacto
- Formulario de contacto (Figura 7)
- Creación de contacto (Figura 7)
- Edición de contacto

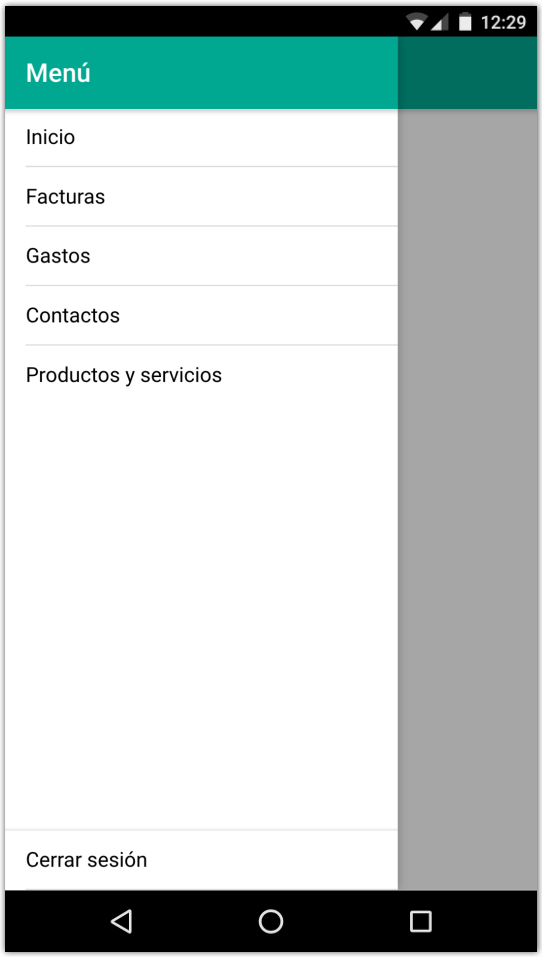


Figura 5. Menú lateral

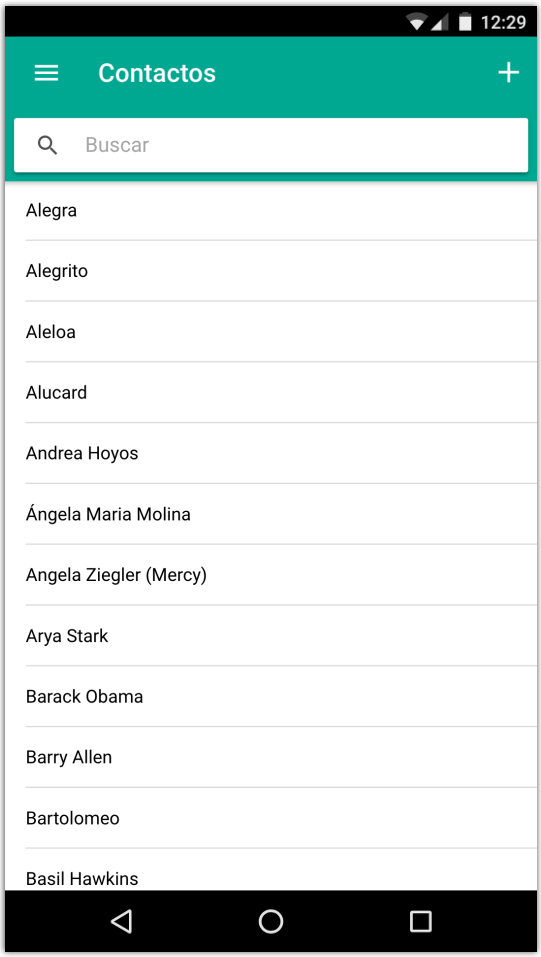


Figura 6. Index de contactos

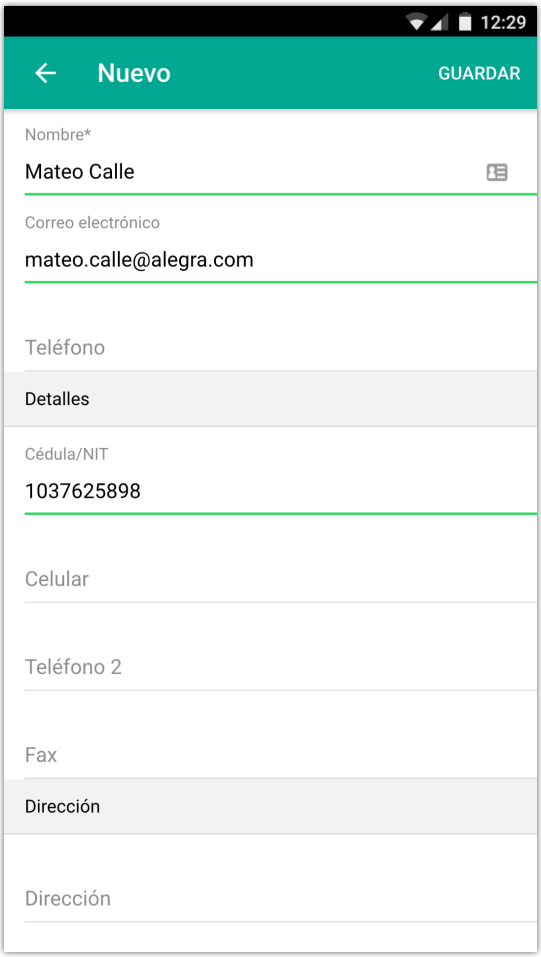


Figura 7. Creación y formulario de contactos

El index de contactos (Figura 6) requería poder buscar contactos por nombre, tener scroll infinito y además poseer la habilidad de hacer *pull-to-refresh* como en Android y iOS. Para las vistas de creación y edición de contacto se creó un componente solamente para el formulario, de modo que la estructura, comportamiento y diseño del formulario quedara en un solo lugar, y así no hubo necesidad de repetir el código a pesar de ser pantallas con diferentes acciones al momento de guardar. De igual forma se incluyó toda la lógica de validación dentro de ese mismo componente.

Sprint 2:

- Index de productos
- Detalle de producto (Figura 8)

- Formulario de producto (Figura 9)
- Creación de producto
- Edición de producto (Figura 9)

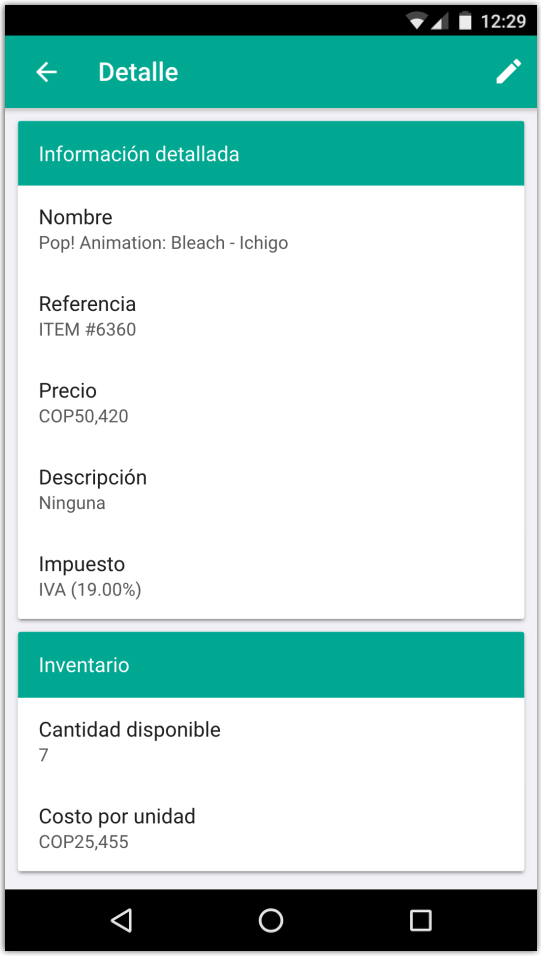


Figura 8. Detalle de producto

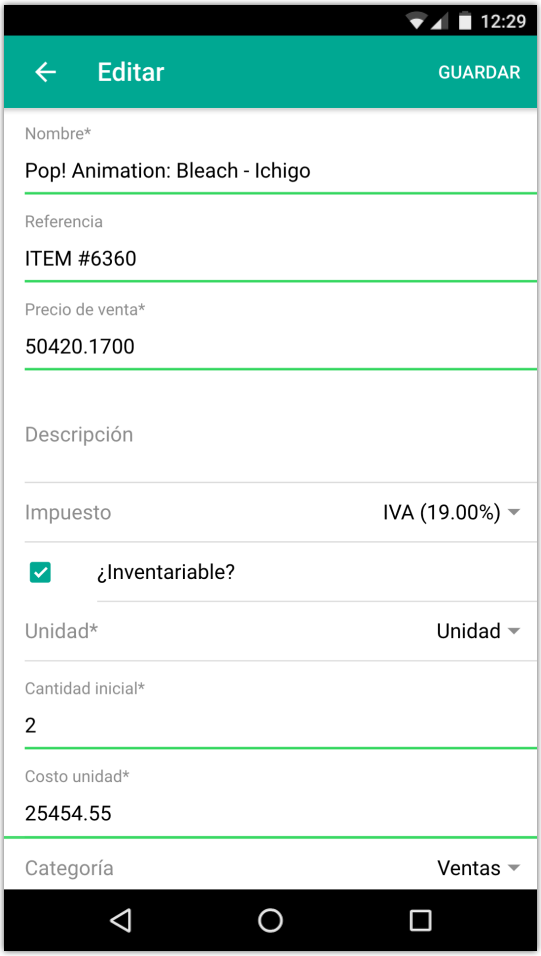


Figura 9. Edición y formulario de producto

Durante este sprint no se encontró ningún inconveniente ya que fue muy parecido al anterior sólo que cambiaba el tipo de entidad (productos en vez de contactos). Incluso el index de productos tenía exactamente los mismos requisitos que el index de contactos.

Sprint 3:

- Index de facturas (Figura 10)
- Detalle de factura
- Formulario de factura (Figura 11)

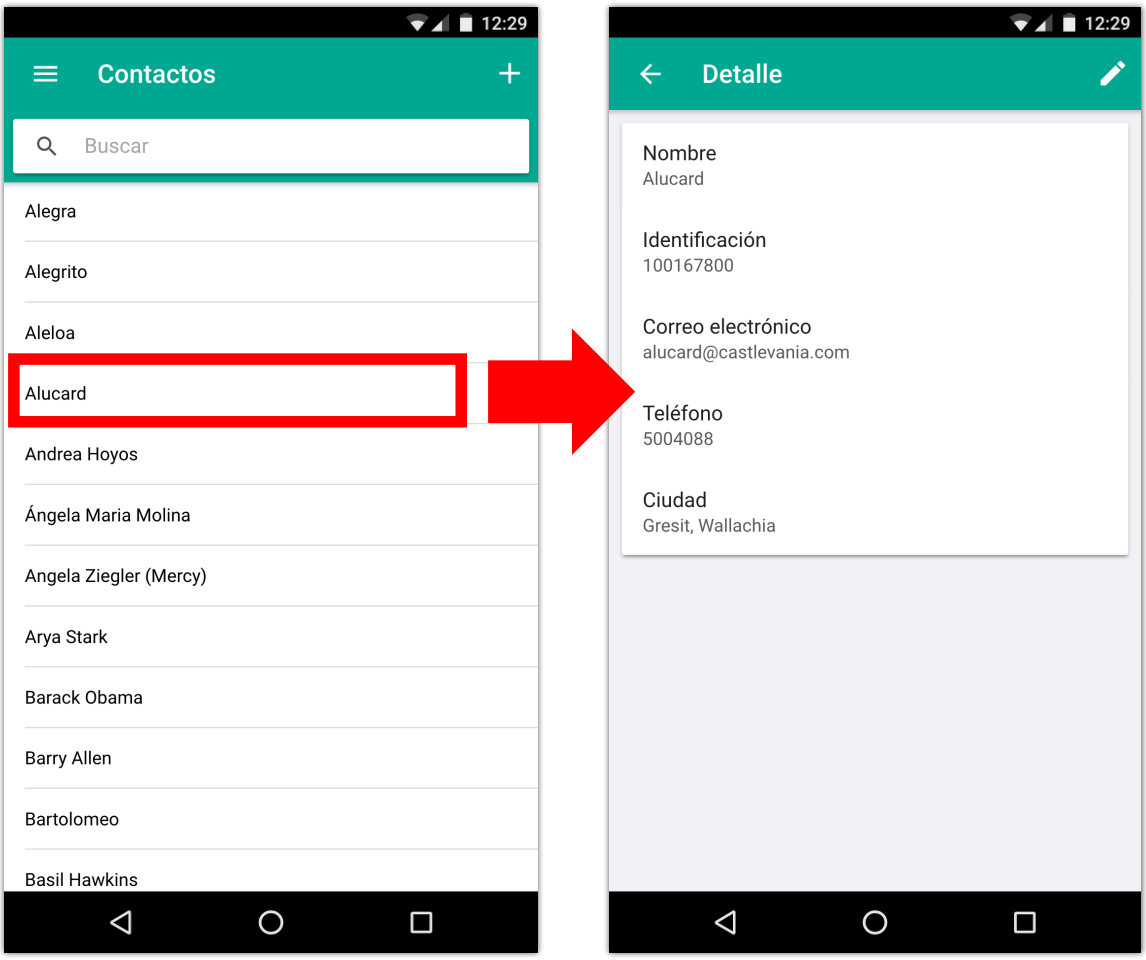
- Emergente de selección de contacto
- Emergente de selección de producto
- Emergente de edición de producto
- Creación de factura (Figura 11)

Figura 10. Index de facturas

Figura 11. Creación y formulario de facturas

Este sprint fue el que más retos presentó debido a la complejidad del modelo de facturas. Para crear una nueva, se tiene como requisito la selección del contacto al que se le asignará la factura, también se deben poder seleccionar productos y editarlos (cantidad, descuento, entre otros) si así se desea, las fechas de creación y vencimiento, etc. Para poder solucionar todos estos retos se obligó a extraer toda la funcionalidad del index de contactos y aislarlo en un nuevo componente de modo que fuera reutilizable, pero que, al momento de seleccionar un contacto, ocurriera una acción diferente dependiendo si se

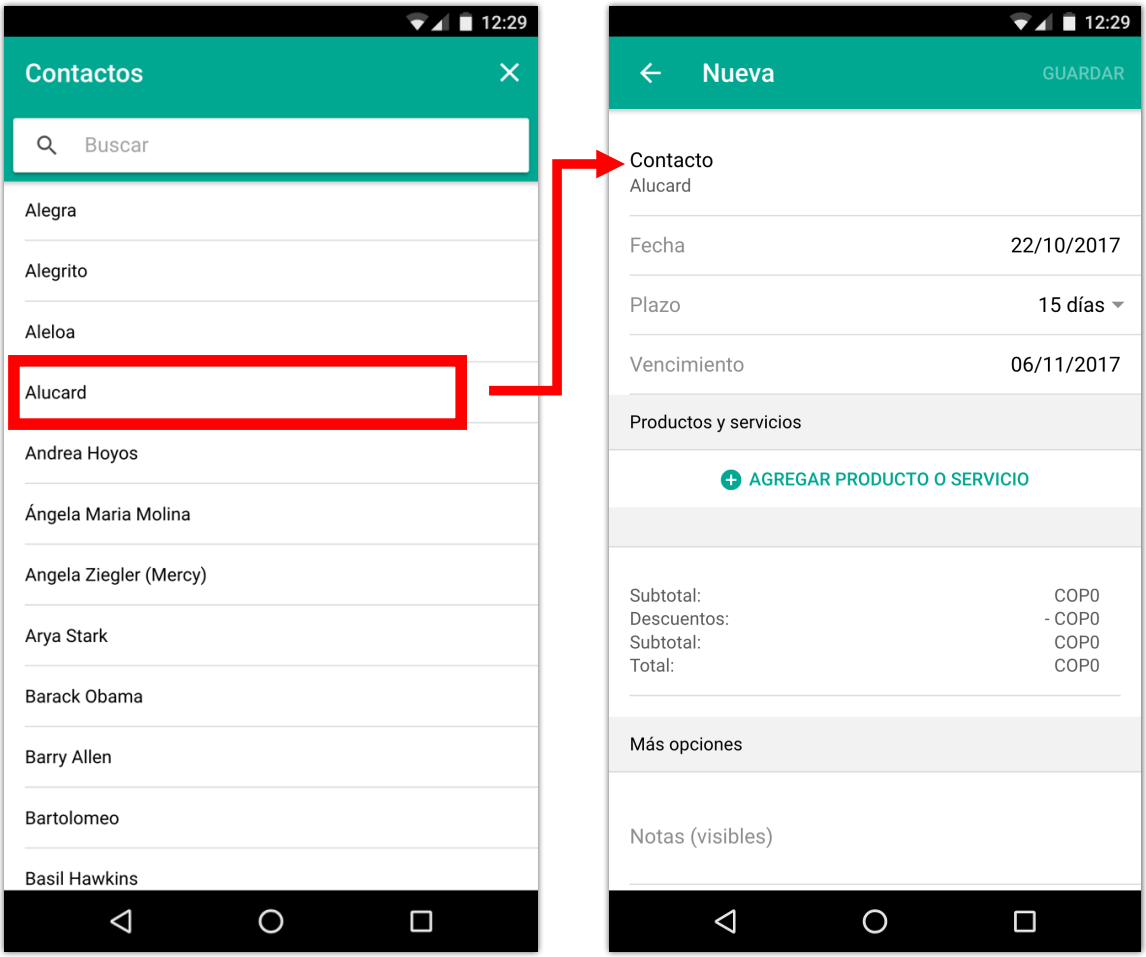
encontraba en el emergente o en el index de contactos. Asimismo, se presentó esta misma situación con los productos.



Index de contactos

Detalle de contacto

En el index de contactos, al seleccionar un item de la lista, se muestra el detalle del contacto seleccionado.



Emergente de contactos Formulario de factura

En el emergente de contactos, al seleccionar un item de la lista, se llena el campo de contacto en el formulario de factura.

Sprint 4:

- Index de gastos
- Detalle de gasto (Figura 12)
- Formulario de gasto (Figura 13)
- Creación de gasto (Figura 13)
- Emergente de selección de categorías (Figura 14)
- Emergente de edición de categoría

- Pantalla de login (Figura 15)

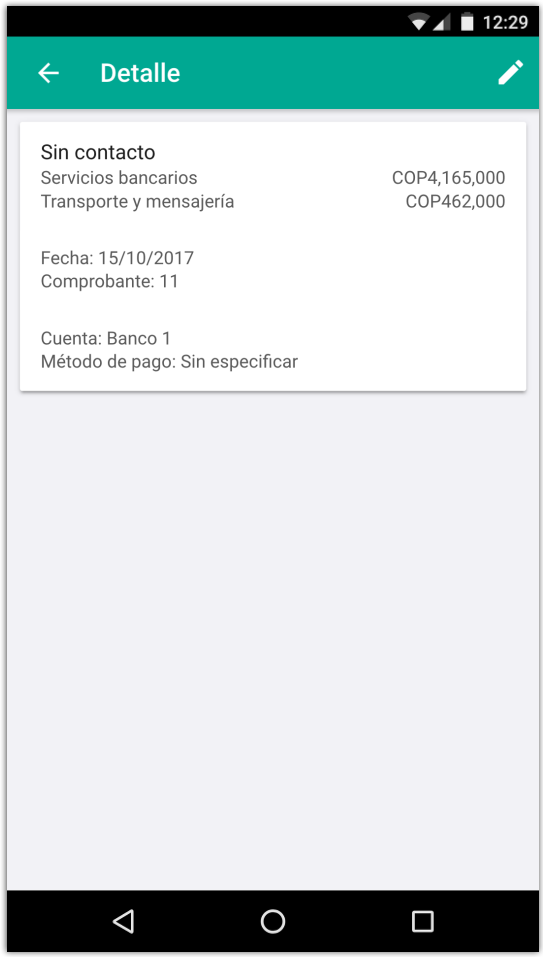


Figura 12. Detalle de gasto

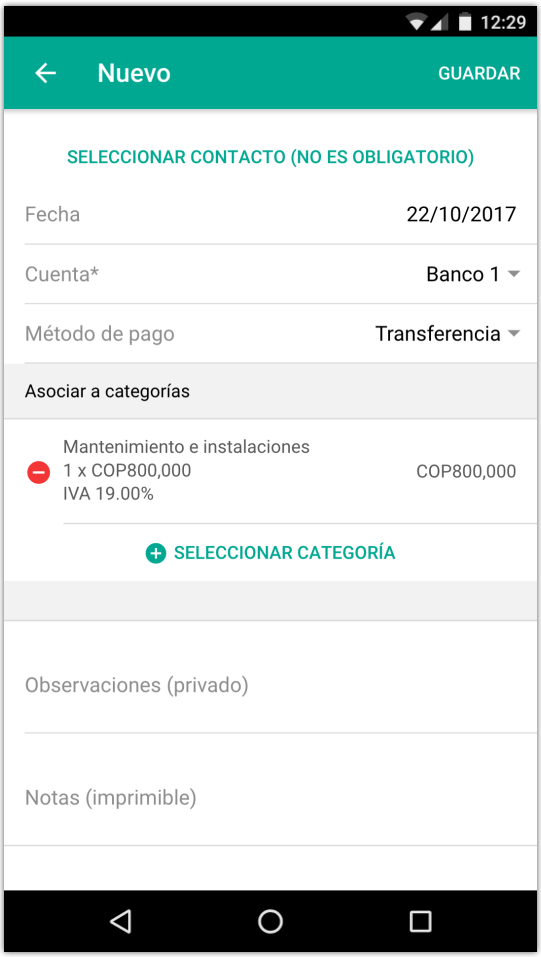


Figura 13. Creación y formulario de gasto

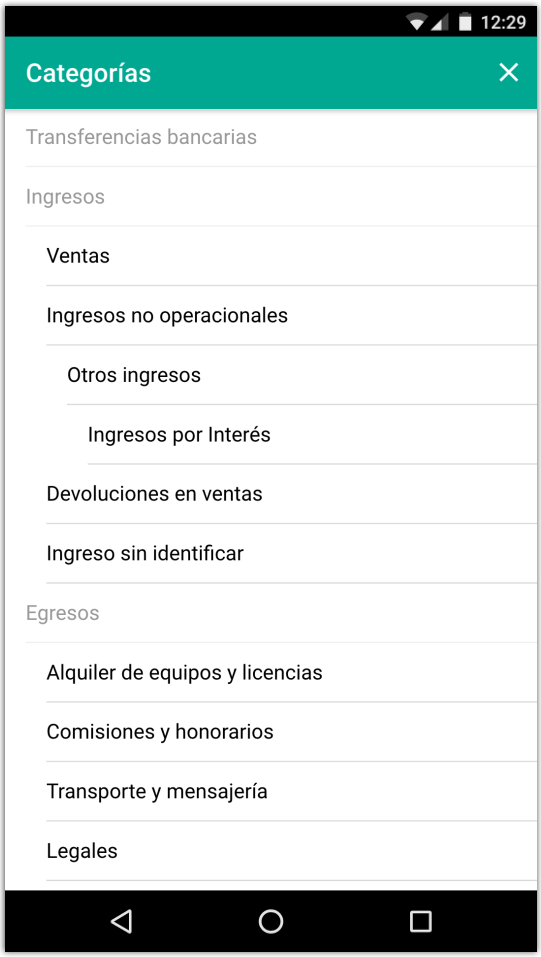


Figura 14. Emergente de categorías

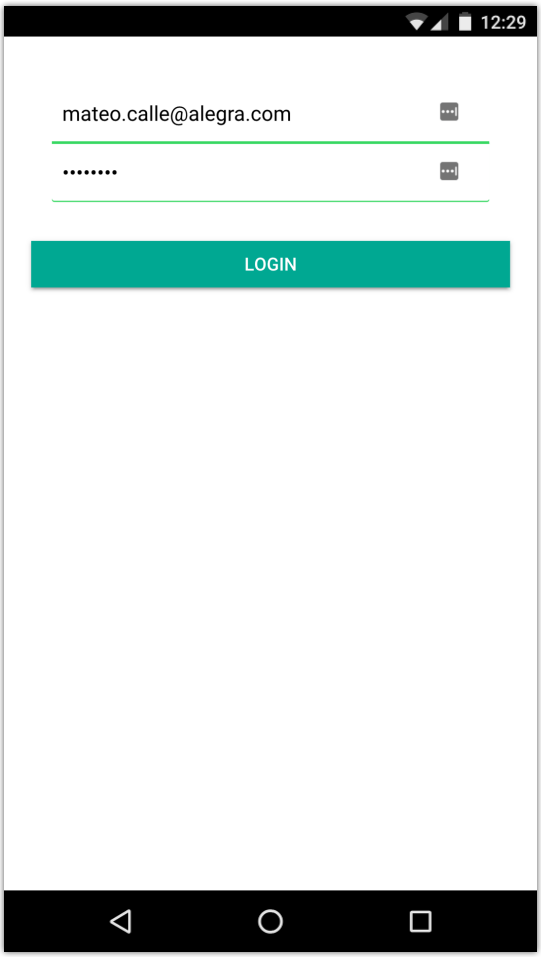


Figura 15. Pantalla de login

En este último sprint no se hallaron casi inconvenientes debido a que varios conceptos eran muy similares al anterior. Sin embargo, el emergente de categorías (Figura 14) trajo algunas complicaciones a causa de su estructura de árbol y la necesidad de crear un componente recursivo.

2.1.4 Comparación funcional

Una vez se terminó la implementación, se procedió a hacer una comparación de rendimiento entre la aplicación actual, y el prototipo desarrollado. A continuación, se muestran los resultados:

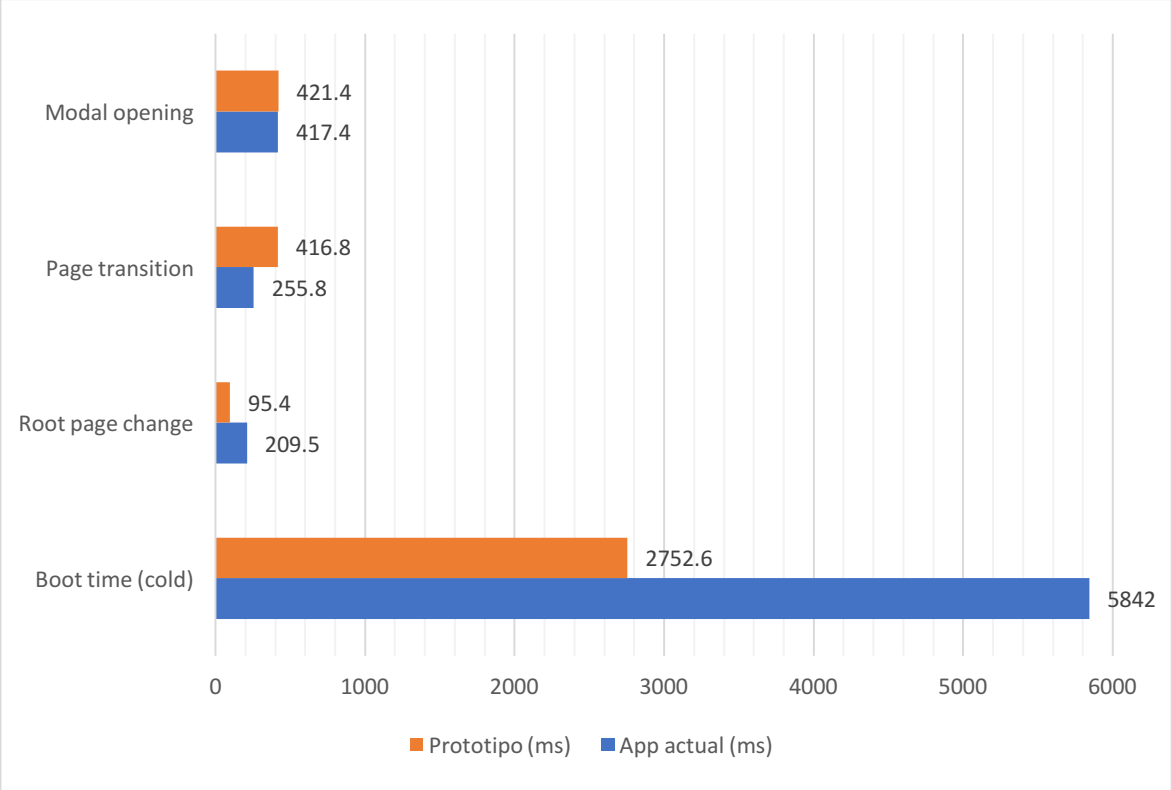


Tabla 1. Comparación de rendimiento entre las diferentes versiones en diversas pruebs (menor es mejor)

Lo primero que se debe decir es que todas las pruebas se hicieron en un equipo Huawei P9 con ninguna otra aplicación abierta. Para cada test, se repitió el proceso unas cinco veces y se sacó el promedio, excepto para el tiempo de inicio, en esta se hicieron 10 pruebas en cada versión debido a que fluctuaba muchísimo más que las otras. Los cálculos se hicieron con la suite de cálculos de rendimiento de JavaScript *`Performance`* (documentación). Para los intervalos de tiempo, se usó *`Performance.measure()`* mientras que el tiempo de inicio se computó con *`Performance.now()`*, que devuelve el período que tardó desde el tiempo de origen.

Los resultados fueron sorpresivos ya que nos fueron tan contundentes como se esperaba. Aunque el prototipo con Ionic 2 si tuvo una gran mejora en unos aspectos, en otros fue mucha más parejo e incluso peor. Para el tiempo de inicio la mejora fue de un 52.9%. Se podría deducir que esto se debe principalmente a que Angular (2+) usa una estrategia *AoT (Ahead of Time)*, que permite pre-compilar las vistas en una etapa de *build* (con Webpack), en vez de que el navegador lo haga al momento de ejecución. Al momento de cambiar la página base, es decir, las del menú lateral, el prototipo fue 114.1ms (54.5%) más rápido en promedio que su contraparte. Se infiere que esto es causa de la navegación totalmente renovada. Ionic 2+ usa un sistema de pila para sus vistas, de modo que, al cambiar la raíz, se reinicia la pila, mientras que Ionic 1 usa un sistema basado en URLs, donde cada vista tiene su propia ruta, probablemente ralentizando el

proceso. En cuanto a la transición entre páginas no raíz (ejemplo: index de facturas al detalle de la seleccionada) fue inesperado ver que el prototipo se demoró casi un 63% más que la aplicación actual. Se teoriza que esto se debe al mismo sistema de navegación por pilas que usa Ionic 2+, de modo que añade la nueva vista encima de la anterior aun guardándola en memoria mientras que Ionic 1 simplemente va a la página seleccionada sin importar qué le pase a la anterior. También se podría atribuir parte del retraso a las animaciones más complejas que usa la nueva versión de Ionic. Por último, la sorpresa más grande fueron los emergentes, ya que cuando se mira desde la experiencia percibida, la aplicación actual se siente lenta y con una animación como entrecortada apenas se abre un emergente, mientras que en el prototipo se aprecia como una acción rápida y fluida, pero a pesar de esto, los datos muestran que prácticamente están parejos. Tal vez las causa de esto es que el tiempo para completar la animación sea el mismo pero la nueva versión de Angular/Ionic tenga mejoras de rendimiento, principalmente en los *frames per second* que logran las animaciones.

En general Angular (2+) mejoró muchísimo la estrategia para de detección de cambio y el algoritmo para manipulación del *DOM*. Esta nueva versión es mucho más eficiente en comparación a su anterior (Peyrott, 2016), pero debido a que estas pruebas fueron relativamente pequeñas, los incrementos de rendimiento no son muy evidenciables. Donde Angular (2+) realmente brilla es cuando se manejan modelos de datos muy grandes y cientos o miles de cambios se hacen en poco tiempo, y asimismo es en estos casos donde AngularJS (1.x) sufre más.

3. PRODUCTOS, RESULTADOS Y ENTREGABLES OBTENIDOS

PRODUCTO ESPERADO	INDICADOR DE CUMPLIMIENTO	OBSERVACIONES (indique si se logró o no su cumplimiento y por qué)	ENTREGABLE
Prototipo de la aplicación móvil.	Todas las funcionalidades básicas deben estar funcionando como deberían.	Cada funcionalidad básica se refiere a lista, detalle y creación de la entidad definida.	Apk de la aplicación móvil listo para correr en un dispositivo Android.
Repositorio con el código fuente del prototipo	El repositorio en la branch *origin/master* debe estar al día con la branch *local/master* donde se hizo el desarrollo del prototipo.	Si la empresa lo desea, se hará transferencia del dueño del repositorio.	Repositorio con el código fuente del proyecto.

Durante la realización del prototipo se vieron evidentes varias ventajas. Las más significativas fueron que Angular (2+) es un framework mucho mejor diseñado y estable, de modo que nunca fue necesario algún tipo de *hack* para lograr implementar lo deseado. Aunque AngularJS (1.x) permitía hacer componentes, la nueva versión fue reescrita desde cero alrededor de esta idea, y es por esto que la reutilización de piezas se facilitó inmensamente y siempre estaba como una prioridad a la hora de pensar en una nueva funcionalidad. Uno de los ejemplos más claros de esto fueron los formularios de contactos y productos, para ambas entidades, las acciones de crear y editar usan el mismo componente y así se mantiene centralizado la validación del modelo. La inclusión de TypeScript se presentó como una herramienta invaluable, tanto para la velocidad y experiencia de desarrollo, como para la facilidad de encontrar errores al momento del desarrollo y no durante la ejecución de la aplicación. Otra ventaja enorme fue la cantidad de componentes nuevos que Ionic 2+ trae por defecto, de modo el desarrollador tiene muchísimas más herramientas disponibles sin tener que hacer uno nuevo personalizado o buscar en línea, y que además cambian de estilos según la plataforma en la que se encuentra experiencia muy cercana a nativa.

Realmente las únicas dificultades que se encontraron fueron la alta curva de aprendizaje que trae Angular (2+) y nuevos conceptos muy poderosos, pero bastante complejos como los *Observables*. Angular es un proyecto muy serio de google con una comunidad muy amplia que ayuda a mantener el ecosistema, por lo que se ve un futuro prometedor para cualquier persona o entidad que se comprometa de lleno a usar este framework. Está

supremamente bien documentado y tiene guías muy completas y comprensivas en su página oficial, y en caso de no encontrar lo que se busca en la página de ellos, nunca fue difícil hallar tutoriales o soluciones cuando había dudas o problemas, siempre estaba entre los tres o cuatro primeros resultados de una búsqueda en Google.

Es así como le queda a Alegra un prototipo muy completo con casos relativamente complejos (como la creación de facturas) y que sirve de base para migrar la aplicación móvil actual a la última versión de Ionic Framework. Aunque sin duda alguna le hace falta muchas funcionalidades que se han ido agregando con el tiempo, las más básicas (facturas, gastos, contactos y productos/servicios) ya se encuentran en el prototipo y pueden servir de fundamento para otras. Además de quedarse con el prototipo, a la empresa le puede servir como un caso de estudio para el desarrollo de nuevos proyectos, como una versión POS exclusiva para tabletas o una modernización de la interfaz gráfica de Alegra tan necesitada de un cambio.

4. CONCLUSIONES Y RECOMENDACIONES

- Se sugiere revisar TypeScript para implementarlo paulatinamente en proyectos ya existentes al interior de la organización.

- Se recomienda migrar la aplicación móvil de Alegra a una nueva versión de Ionic Framework debido a sus inmensas mejorías, o si es el caso, analizar nuevos proyectos como React Native.

- Se invita a la empresa a darle una revisión detallada de Angular (2+) cuando se trata de proyectos muy grandes de frontend. En esos casos de uso es donde el framework realmente brilla.

5. REFERENCIAS

Abed, R. (s.f.). *Hybrid vs Native Mobile Apps – The Answer is Clear*. Recuperado el 8 de Abril de 2016, de Y Media Labs: https://info.dynatrace.com/rs/compuware/images/Mobile_App_Survey_Report.pdf

Fischer, P. (15 de 05 de 2015). *Blogs@Intel*. Recuperado el 20 de 05 de 2016, de Build High-Performance HTML5 Cordova Apps with Crosswalk: https://blogs.intel.com/evangelists/2015/05/12/build-high-performance-html5-cordova-apps-with-crosswalk/

Google. (s.f.). *Android developers*. Recuperado el 20 de 05 de 2016, de WebView: https://developer.android.com/reference/android/webkit/WebView.html

Google. (s.f.). *Google Play*. Recuperado el 20 de 05 de 2016, de ChefSteps: https://play.google.com/store/apps/details?id=com.chefsteps.mobile&hl=en

Google. (s.f.). *Google Play*. Recuperado el 20 de 05 de 2016, de Alegra: https://play.google.com/store/apps/details?id=co.alegra.app

Google. (s.f.). *Micro-Moments*. Recuperado el 8 de Abril de 2016, de Think with Google: https://www.thinkwithgoogle.com/collections/micromoments.html

Gundersen, M. (2013). A comparison of the two-way binding in AngularJS, EmberJS and KnockoutJS. Berlín, Alemania.

Katie. (10 de 09 de 2014). *The official Ionic blog*. Recuperado el 20 de 05 de 2016, de Built with Ionic: Sworkit: http://blog.ionic.io/built-with-ionic-sworkit/

Katie. (14 de 01 de 2016). *The official Ionic blog*. Recuperado el 20 de 05 de 2016, de Built with Ionic: ChefSteps: http://blog.ionic.io/built-with-ionic-chefsteps/

Lynch, M. (05 de 01 de 2016). *The official Ionic blog*. Recuperado el 20 de 05 de 2016, de How 2015 Went for Ionic: http://blog.ionic.io/how-2015-went-for-ionic/

Lynch, M. (9 de January de 2017). *Ionic's 2016 - By the Numbers*. Obtenido de Medium: https://medium.com/ionic-and-the-mobile-web/ionics-2016-by-the-numbers-3a8e2c65177b

Perez, S. (21 de Agosto de 2014). *Majority Of Digital Media Consumption Now Takes Place In Mobile Apps*. Recuperado el 8 de Abril de 2016, de Techcrunch: http://techcrunch.com/2014/08/21/majority-of-digital-media-consumption-now-takes-place-in-mobile-apps/

The Apache Software Foundation. (s.f.). *Apache Cordova*. Recuperado el 20 de 05 de 2016, de Plugins: https://cordova.apache.org/docs/en/latest/guide/overview/#plugins

Yassour, Y. (27 de September de 2017). *Built with Ionic: Microsoft Flow app*. Recuperado el 3 de October de 2017, de Ionic Blog: http://blog.ionic.io/built-with-ionic-microsoft-flow-app/